LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-403930

PARALLEL IMPLEMENTATION OF THE TOPAZ OPACITY CODE: ISSUES IN LOAD-BALANCING

V. Sonnad, C. A. Iglesias

May 15, 2008

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

PARALLEL IMPLEMENTATION OF THE TOPAZ OPACITY CODE: ISSUES IN LOAD-BALANCING

VIJAY SONNAD
CAR-DEPCom

and

CARLOS A. IGLESIAS
PS-High Energy Density Physics (V Division)

ABSTRACT

The TOPAZ opacity code explicitly includes configuration term structure in the calculation of bound-bound radiative transitions. This approach involves myriad spectral lines and requires the large computational capabilities of parallel processing computers. It is important, however, to make use of these resources efficiently. For example, an increase in the number of processors should yield a comparable reduction in computational time. This proportional "speedup" indicates that very large problems can be addressed with massively parallel computers. Opacity codes can readily take advantage of parallel architecture since many intermediate calculations are independent. On the other hand, since the different tasks entail significantly disparate computational effort, load-balancing issues emerge so that parallel efficiency does not occur naturally. Several schemes to distribute the labor among processors are discussed.

1. INTRODUCTION

The TOPAZ code was developed to calculate the opacities of mid-Z elements. The three major contributors to photon absorption are bound-bound, bound-free, and free-free transitions. [1] For mid-Z elements, the opacity from the bound-bound process involves contributions from a large number of transition arrays. Each array consists of radiative transitions involving pairs of initial and final electronic configurations. In turn, each configuration has a number of total angular momentum terms resulting from the angular momentum couplings of the individual electrons in the configuration. Finally, all possible transitions between each term in the initial and final configurations must be considered; hence, the terminology detailed term accounting (DTA) method. [2]

The calculations for a transition array start by solving for the eigenvalues and eigenvectors of the Hamiltonian corresponding to the initial and final configurations. Then follows a pre and post multiplication of the resulting eigenvectors by the transition matrix. [2] The matrix elements are made up of radial and angular expressions. After careful evaluation, the ANCO code was chosen for the calculation of the angular contribution to the matrix elements. [3,4]

While the angular parts are unique to each configuration term, the radial contributions are unchanged for all terms corresponding to a given relativistic configuration. Nevertheless, for complex configurations, the overall costs of a transition array calculation are large compared to the radial calculations alone, and there is no significant penalty in recalculating the radial expressions as required.

The DTA approach can generate myriad transitions so that these are large calculations; thus, parallel processing is necessary to keep the computations tractable. The number of terms depends on the complexity of the configuration; for example, the number of open shells. [2] This leads to wildly disparate computational times for transitions involving different initial-final relativistic configuration pairs. Consequently, a naïve parallel implementation can lead to very poor scaling due to load imbalance among the tasks. The present work describes the structure of the computations for bound-bound transitions and the different approaches that have been developed in order to obtain good parallel performance.

2. PARALLEL IMPLEMENTATION

Given the structure for the bound-bound computations, it was decided to employ a manager-worker approach as the most appropriate for parallel implementation. The basic strategy is

simple: of the available processors, one is termed a manager processor with the rest being worker processors. The manager processor initially assigns a task to each available worker processor. When any worker processor completes its assigned task, the manager collects the individual worker-processor results and assigns a new task to that worker. The process is repeated until all available tasks are assigned and completed. In the final stage the manager processor uses the compiled results to calculate a variety of quantities not requiring much computational effort such as the Rosseland mean opacity. [1]

There are several challenges to realizing high parallel processing efficiency with a manager-worker approach:

i. How are independent tasks to be computed in parallel by the worker processors defined?
ii. Each task must be complex enough so that the communication time between manager and worker is small compared to the computation time for that task.
iii. Simultaneously, it is desirable to have many tasks to take advantage of a large number of processors.
iv. The tasks should be comparable in computation times to avoid load-balancing problems.

Each of these is considered in turn:

i. It would be convenient to define an independent task as an initial and final relativistic configuration pair (RCP). Although the calculation for each RCP is independent of all others, a redundancy is introduced since the radial integrals are the same for several relativistic configurations associated with a single non-relativistic configuration. It was already noted that for complex configurations the radial contributions only constitute a small fraction of the computational effort. Therefore, when grouping RCP's, the low cost of repeating the radial calculations is neglected. With this consideration, a task is defined as a group of RCP's.
ii. From experience it is known that for relatively simple RCP's the communication time is a considerable fraction of the computation time for that task. Accordingly, the flexibility to group from one to many RCP's is the right approach to meet this requirement.
iii. It follows that assigning each individual RCP as a task would maximize their number. This, however, leads to small tasks that conflict with requirement ii) with poor parallel efficiency due to communication costs.

iv. The issue of load balancing is crucial to obtaining satisfactory parallel performance when the communication costs are small. Even if requirements i) through iii) are satisfied, there remains the relative computational costs of each of the tasks. If there are a few tasks that are much more expensive than the others, this can lead to many idle processors while a few are loaded with the expensive tasks. It is desirable to have tasks where the distribution of computational times does not vary greatly.

The parallel performance issues are strongly related to the grouping of the RCP's into tasks. Unfortunately, there is no obvious a priori way for determining an optimal scheme. In the following sections different groupings and the resulting parallel efficiencies are described.

2.1 Grouping by Non-Relativistic Configurations

The input to the TOPAZ code consists of list of initial non-relativistic configurations (NRC's). This permits the possibility of using *LS* coupling for light elements. [2] For heavier elements requiring *jj* coupling, each NRC is expanded into all possible relativistic configurations. [2] It is natural to consider defining a single task by grouping all RCP's associated with a single NRC. Conveniently this also avoids the redundancy with the radial calculations.

Results for TOPAZ calculations with 808 initial non-relativistic configurations using the NRC grouping with varying number of total processors are presented in Fig. 1. The figure shows very poor scaling with essentially constant speedup beyond 32 processors. The reason becomes evident by examining the time required for each task. These times are displayed in Fig. 2 and shows a wide variation in the times required for each task, with the most expensive as much as three orders of magnitude larger than the least expensive. A single expensive task could therefore occupy a processor for the entire duration of the run and the addition of more processors cannot reduce the time required to complete the run. In this case, the poor parallel performance is entirely due to poor load balancing.

2.2 Distributed Grouping for NRC

An obvious solution to better load balancing is to reduce the time for the most expensive tasks by distributing the work among several processors. It is possible to break up the NRC tasks into smaller groups of RCP's. At the same time, it is not efficient to break up inexpensive NRC tasks since that would entail unnecessary communication overhead. Thus, one needs the capability to distribute the expensive NRC tasks on several processors while keeping the less expensive ones on a single processor.

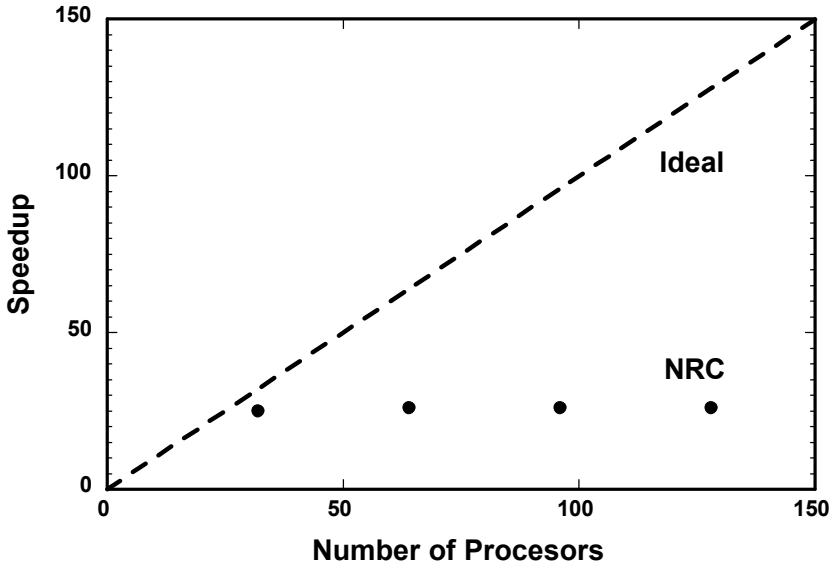

Fig. 1 Speedup as a function of processors for NRC grouping.

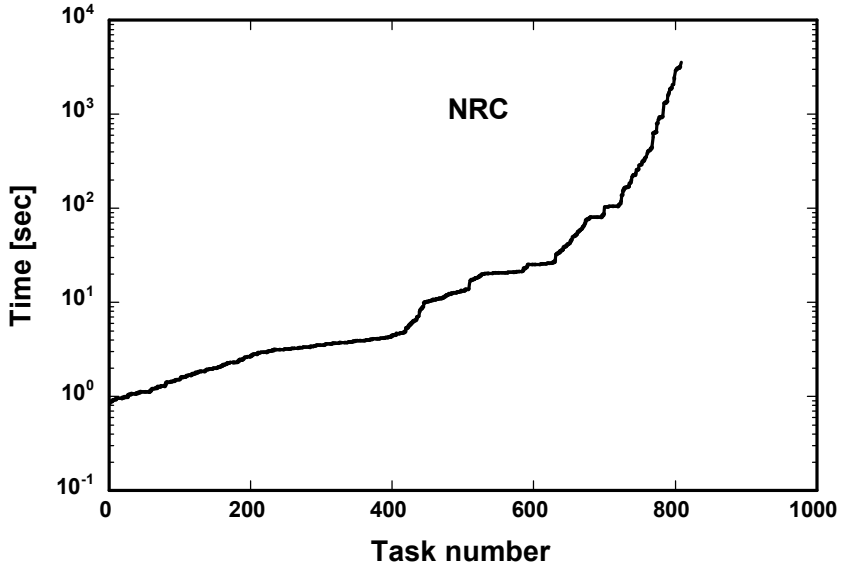

Fig. 2 Task computational times for NRC grouping in Fig. 1.

This can be implemented by a two-level parallel scheme where the manager processor communicates the information associated with an expensive task to a sub-manager processor which then farms out the individual RCP's to worker processors. Inexpensive NRC tasks on the other hand, are sent directly from the manager processor to a single worker processor. This two-level approach shall be denoted as "distributed non-relativistic configurations" (DNRC) and is illustrated schematically in Fig. 3.

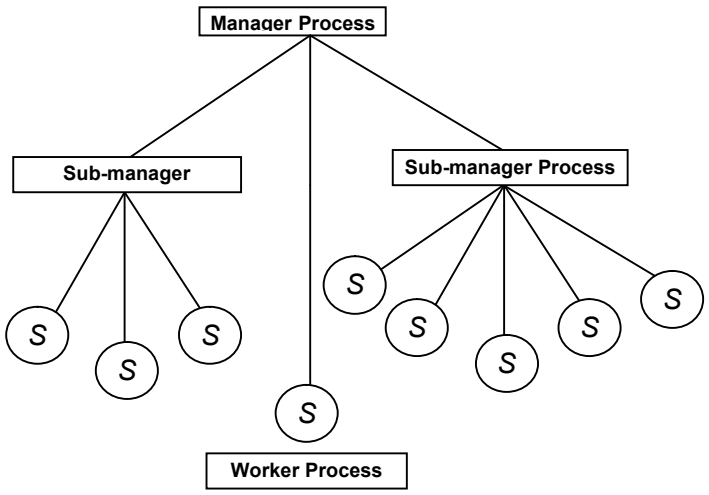

Fig. 3 Schematic illustration of DNRC grouping of RCP's where the worker processors are denoted as S1, S2, and so on.

In order to implement the DNRC method it is necessary to distinguish between an expensive and an inexpensive NRC task. It is possible to show that there is a strong correlation between the computational time and the number of relativistic initial-final configuration pairs associated with an initial non-relativistic configuration. It is straightforward and computationally fast to obtain this information from the list of initial non-relativistic configurations before performing the actual calculations. The input list can then be approximately ordered starting with the most expensive to the least expensive along with the added information concerning RPC's. Every task is then sent from the manager to a sub-manager or a worker processor as appropriate.

Two variations of the DNRC are implemented for the example with 808 NRC tasks in Fig. 1. The first approach, DNRC-V, uses a smoothly varying distribution in the number of processors for each configuration ranging from 31 for the most expensive to 1 for the least expensive. The

second approach, DNRC-8, fixes the number of processors at 8 for the most expensive 142 NRC's and 1 for the remaining 666.

The two-level parallel approach has the additional complication that a given task may require more processors than are available at the time. In this case the procedure dispatches the most expensive task that requires the available processors. With carefully constructed data structures, this decision can be made in constant time regardless of the total number of processors or tasks, and the overhead on the manager processor is minimal.

It is instructive to plot in Fig 4 the distribution of task times for the two approaches. Clearly, DNRC-V has greatly reduced the time for the most expensive task over DNRC-8; hence, it is reasonable to expect improved load balancing for DNRC-V. In Fig. 5 is plotted the speedup for the various task methods and shows that both DNRC schemes improve on the NRC approach with DNRC-V the most parallel efficient.

The speedup behaviors are interesting in that DNRC-8 is better for small number of processors while DNRC-V scales better with a larger number of processors. This is readily explained by considering the load balancing within a sub-manager. For a single task, using a smaller number of processors as in DNRC-8 as compared to DNRC-V, gives better parallel speedup for a sub-manager. For a smaller total number of processors, the overall load balance is comparable with DNRC-8 and DNRC-V; hence, the overall parallel speedup is better with DNRC-8. This advantage, however, is lost when using a larger number of processors because the overall load balancing is much better with DNRC-V, which overwhelms the effect of the better efficiency of DNRC-8 within the sub-manager.

As expected, having several processors working on expensive tasks leads to dramatic gains in efficiency and scalability over the NRC approach. Although possible, it is not in general easy to pick an optimal assignment of processors to each NRC task beforehand and this can significantly affect the parallel efficiency. An approach that provides high efficiency without having such detailed information in advance is discussed in the next section.

2.3 Grouping by relativistic configurations

In the approaches adopted above, the basic task consisted of all relativistic pairs belonging to a single initial non-relativistic configuration. The next approach emphasizes the initial relativistic configuration or RC. All pairs with the same initial relativistic configuration are grouped together and assigned as a single task to a processor. This scheme returns to single level

parallelism, one manager plus workers, and avoids optimization problems assigning processors to the NRC's. Displayed in Fig 4 are the distributions of computing times for tasks grouped by RC. While there is again load imbalance, there are now many more tasks and this leads to the parallel performance shown Figs 5 and 6.

The RC approach performs well with relatively few processors (relative to the number of tasks); however, when there are a large number of processors, DNRC-V performs best. This is because even modest load balancing effects become critical with a large number of processors, and an approach (such as DNRC-V) which is globally load balanced does well even though the same method performs poorly with few processors. Since real TOPAZ calculations almost always involve many more tasks than processors, the RC scheme is the default approach.

The RC grouping has additional advantages. Firstly, it results in a cleaner and simpler implementation. This reduces the number of potential errors and future modifications or refinements can be more easily incorporated. Secondly, it is possible to save part of the calculations of the first initial-final relativistic configuration for all remaining pairs within the RC group. Examples are the eigenvalues and eigenvectors for the initial relativistic configuration. Also since all the final relativistic configurations are related, it is in principle although not yet implemented, possible to save part of the final relativistic calculation.

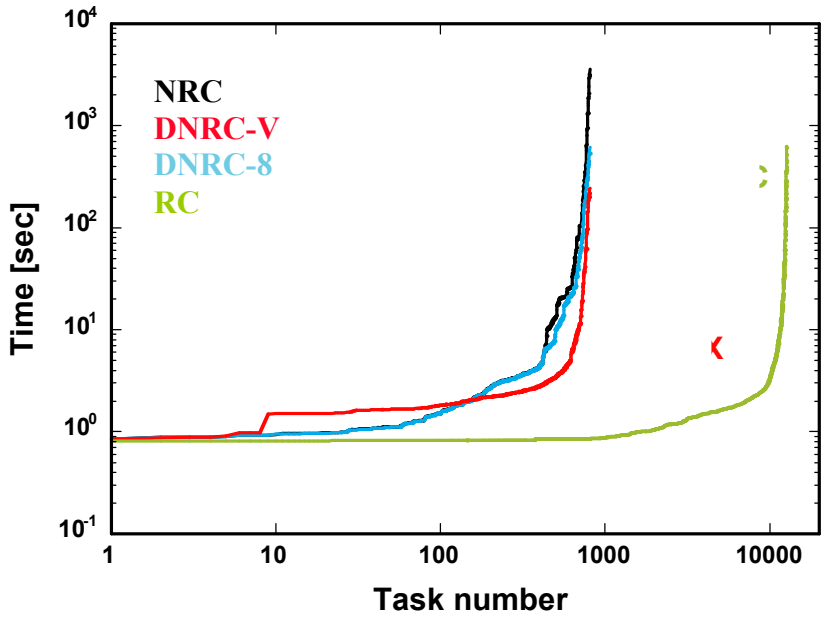

Fig. 4 Distribution of task times for all 4 grouping scheme approaches as a function of task numbers.

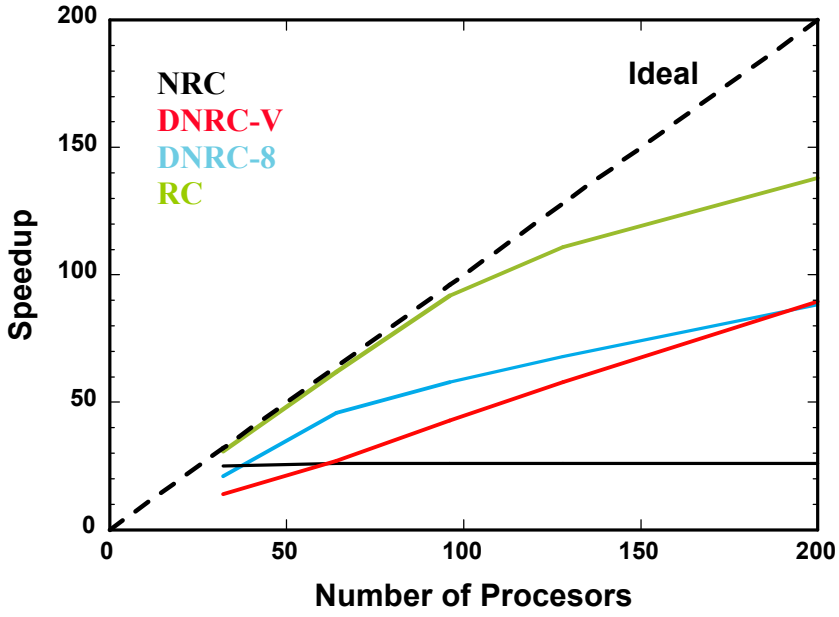

Fig. 5 Speedup as a function of processors for all grouping schemes with relatively few processors.

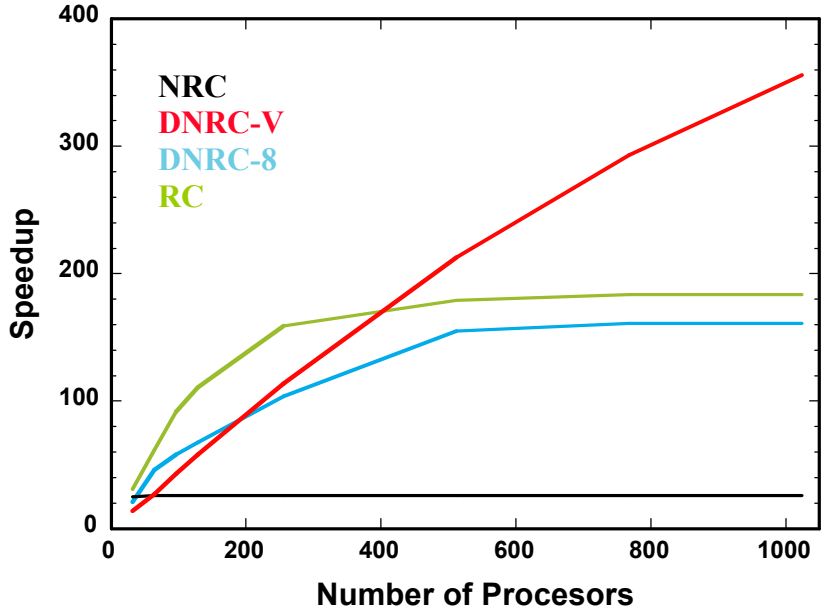

Fig. 6 Speedup as a function of processors for all grouping schemes with a larger number of processors.

3. CONCLUSIONS

The parallel efficiency of the TOPAZ code is entirely determined by the disparities in computation times of the individual tasks. When a single processor is assigned to a task composed of an initial non-relativistic configuration (NRC), the parallel scaling is very poor. Two methods have been described to improve load balancing between tasks.

In the first, the grouping of relativistic configuration pairs is as before, but several processors are assigned to the most expensive groups. The optimal number of processors is not known before performing the calculations so two different methods of allocating processors were attempted. One is a smoothly varying distribution of processors proportional to the number of pairs (DNRC-V) and the other is a fixed number of processors for the most expensive configurations and a single processor for each of the remaining configurations (DNRC-8). This was found to give a significant improvement over the NRC approach, but it is very sensitive to the choice for the number of processors. In the second approach, the grouping of initial-final relativistic configuration pairs was changed. The new organization consists of assigning to the same processor all pairs with the same initial relativistic configuration.

It turns out that the DNRC-V method performs best when the number of processors is large in comparison to the number of initial non-relativistic configurations and the RC method works best for a lesser number of processors. It is tempting to combine the two methods to obtain one that would perform well for both small and large number of processors. It should also be noted, however, that the good performance of the DNRC-V depends on optimizing the number of processor. In the example presented here that was approximately accomplished by trial and error, but this is obviously not possible in the general case. Also note that there are constraints from the radial computations that would increase in significance with reducing total time for each task; thus, it would be eventually inefficient to re-compute them as needed.

ACKNOWLEDGEMENTS

The authors would like to thank G. Gaigalas and S. Fritzsche for their kind assistance in assuring the correctness of the angular momentum calculations with the ANCO package. Also Mal Kalos has provided many valuable suggestions about schemes for parallel implementation. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory in part under Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344.

REFERENCES

[1] J.P. Cox & R.T. Giuli, *Principles of Stellar Structure* (Gordon & Breach, New York, 1968)
[2] R. Cowan, *The Theory of Atomic Structure and Spectra* (University of California Press, Berkeley, 1972)
[3] G. Gaigalas & S. Fritzsche, & I.P.Grant, *CPC* **139**, 263(2001)
[4] G. Gaigalas & S. Fritzsche, *CPC* **148**, 349(2002)